Exploratory Data Analysis of Google Play Store Apps

Objective:
This analysis explores the Google Play Store dataset, examining app characteristics, user engagement, and sentiment expressed in reviews.

Scope:

- Identify top app categories and their popularity
- Compare Free vs Paid apps and pricing trends
- Examine app ratings and review distributions
- Analyze user sentiment using VADER
- Highlight key insights with charts and visualizations

Tools: Python, Pandas, NumPy, Matplotlib, Seaborn, VADER Sentiment Analyzer, WordCloud

Step 1 — Upload Dataset

To begin the analysis, upload the following files:

- `googleplaystore.csv` — Contains app metadata (category, rating, installs, price, type, etc.)
- `googleplaystore_user_reviews.csv` — Contains user reviews for the apps

Instructions:

- Click **Run** on the next cell and select the files from your computer.
- If files are already placed in `/mnt/data`, the notebook will locate them automatically.

```
# Run this to upload files
from google.colab import files
uploaded = files.upload()
```

```
Choose Files 2 files
googleplaystore.csv(text/csv) - 1360155 bytes, last modified: 9/11/2025 - 100% done
googleplaystore_user_reviews.csv(text/csv) - 7669276 bytes, last modified: 9/11/2025 - 100% done
Saving googleplaystore.csv to googleplaystore (1).csv
Saving googleplaystore_user_reviews.csv to googleplaystore_user_reviews (2).csv
```

Step 2 — Load and Clean Data

In this step:

1. Load the app and review datasets into Pandas DataFrames.
2. Clean and preprocess columns:
   - Convert `Installs`, `Price`, `Reviews`, `Rating` to numeric types
   - Handle missing values and duplicates
3. Prepare the datasets for analysis, ensuring consistency for merging reviews and app metadata.

```
# Load + clean basic
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (9,5)

# locate files (works if uploaded via files.upload() or placed in /mnt/data)
candidates = [
  "googleplaystore.csv",
  "googleplaystore_user_reviews.csv",
  "/mnt/data/googleplaystore.csv",
  "/mnt/data/googleplaystore_user_reviews.csv"
]

apps_file = None
reviews_file = None
for p in candidates:
    if os.path.exists(p):
        if "user_reviews" in p or "reviews" in p and "googleplaystore" in p and "user" in p:
            reviews_file = p
        elif "googleplaystore" in p and "user" not in p:
            apps_file = p

# If names from upload are different, try to pick by filename pattern
if apps_file is None:
    for f in os.listdir():
        if "googleplaystore" in f and "user" not in f and f.endswith(".csv"):
            apps_file = f
if reviews_file is None:
    for f in os.listdir():
        if "review" in f.lower() and f.endswith(".csv"):
            reviews_file = f

if apps_file is None:
    raise FileNotFoundError("Cannot find googleplaystore.csv. Upload it (use earlier upload cell) or place it in /mnt/data.")
print("Apps file:", apps_file)
print("Reviews file:", reviews_file if reviews_file else "(none found)")

def safe_read(path):
    try:
        return pd.read_csv(path)
    except Exception:
        return pd.read_csv(path, encoding="latin-1")

apps = safe_read(apps_file)
# basic cleaning apps
apps.columns = apps.columns.str.strip()
apps.drop_duplicates(inplace=True)
# Numeric conversions
if "Reviews" in apps.columns:
    apps["Reviews"] = pd.to_numeric(apps["Reviews"], errors="coerce")
if "Installs" in apps.columns:
    apps["Installs"] = apps["Installs"].astype(str).str.replace("+","",regex=False).str.replace(",","",regex=False)
    apps["Installs"] = pd.to_numeric(apps["Installs"], errors="coerce")
if "Price" in apps.columns:
    apps["Price"] = apps["Price"].astype(str).str.replace("$","",regex=False).replace("Free","0",regex=False)
    apps["Price"] = pd.to_numeric(apps["Price"], errors="coerce").fillna(0)
if "Rating" in apps.columns:
    apps["Rating"] = pd.to_numeric(apps["Rating"], errors="coerce")
apps = apps.dropna(subset=["App"]).reset_index(drop=True)

# Load reviews if present
reviews = pd.DataFrame()
if reviews_file:
    reviews = safe_read(reviews_file)
    reviews.columns = reviews.columns.str.strip()
    # find likely text column
    for c in ["Translated_Review","Review","review","Translated Review","Content","Review Text"]:
        if c in reviews.columns:
            text_col = c
            break
    else:
        # fallback to second column or first
        text_col = reviews.columns[1] if reviews.shape[1]>1 else reviews.columns[0]
    reviews[text_col] = reviews[text_col].astype(str).fillna("").str.strip()
    reviews = reviews[reviews[text_col].str.len()>0].copy()
    reviews = reviews.rename(columns={text_col: "clean_review"})
    reviews["clean_review"] = reviews["clean_review"].astype(str)
    print("Loaded reviews. Sample column used:", "clean_review")
else:
    print("Reviews file not found; we'll do apps-only analysis.")

print("Apps shape:", apps.shape)
print("Reviews shape:", reviews.shape)
apps.head()
```

```
Apps file: googleplaystore.csv
Reviews file: googleplaystore_user_reviews.csv
Loaded reviews. Sample column used: clean_review
Apps shape: (10358, 13)
Reviews shape: (64295, 5)
```

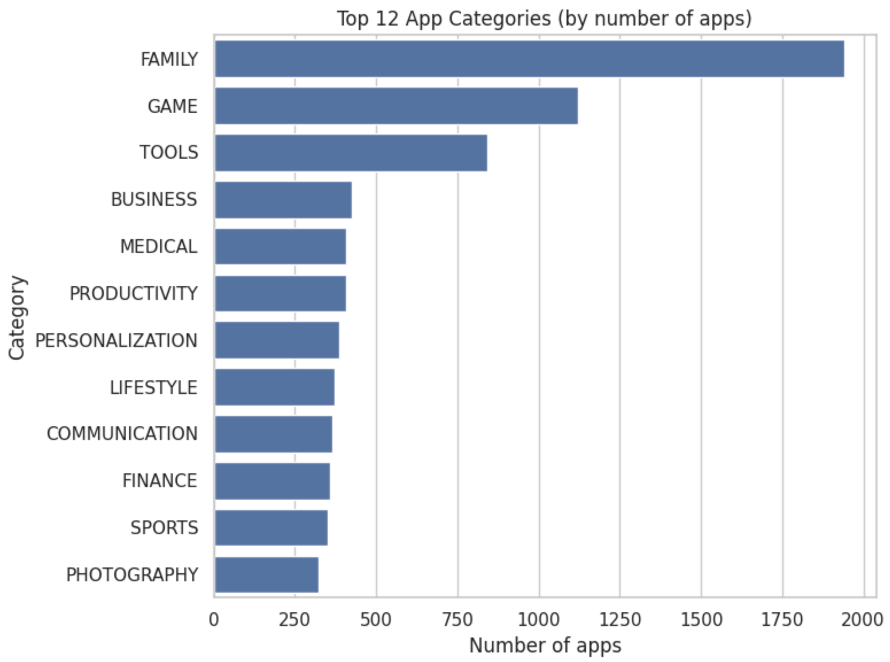
	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159.0	19M	10000.0	Free	0.0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967.0	14M	500000.0	Free	0.0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510.0	8.7M	5000000.0	Free	0.0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644.0	25M	50000000.0	Free	0.0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up

Next steps: Generate code with apps | View recommended plots | New interactive sheet

Insight 1 — Top App Categories

This chart identifies the categories with the largest number of apps published on the Play Store.
It highlights where developers focus their efforts and provides an overview of market trends.

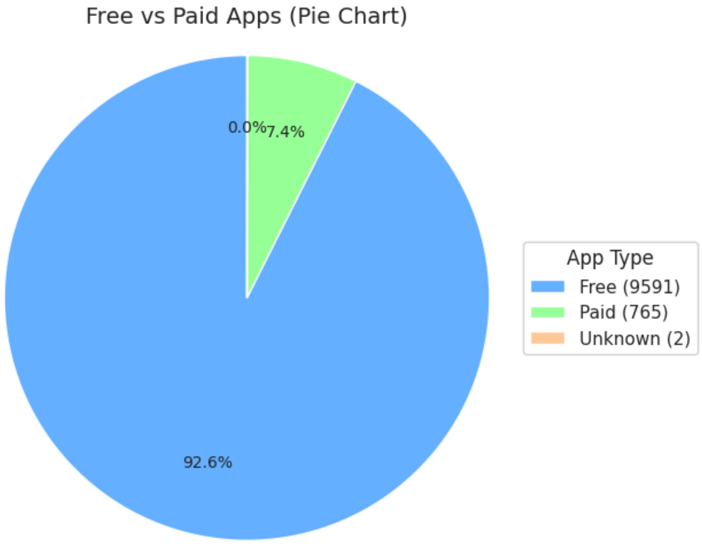
```
# Top 12 categories by app count
top_cat = apps["Category"].value_counts().head(12)
plt.figure(figsize=(8,6))
sns.barplot(x=top_cat.values, y=top_cat.index)
plt.title("Top 12 App Categories (by number of apps)")
plt.xlabel("Number of apps")
plt.tight_layout()
plt.show()
```



Insight 2 — Free vs Paid Apps

This visualization compares the number of Free and Paid apps.
It shows the proportion of apps monetized through pricing versus those available at no cost, providing insight into market strategy.

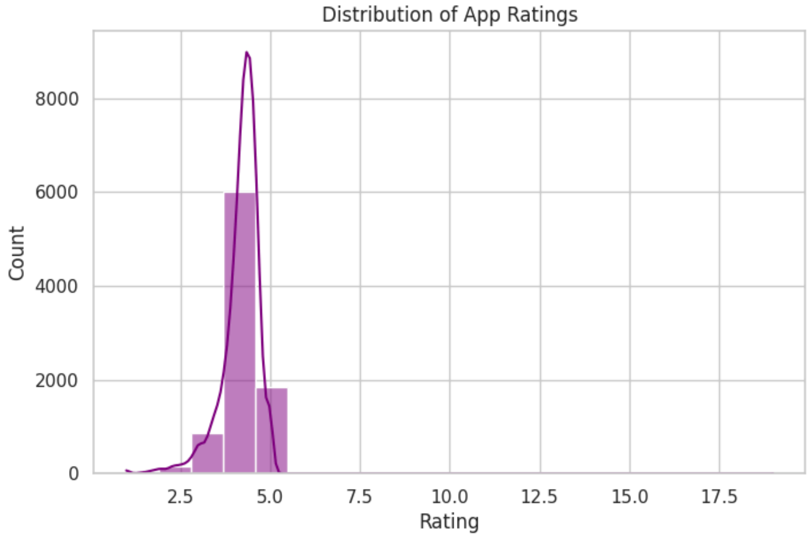
```
apps["Type"] = apps["Type"].replace("0","Unknown").fillna("Unknown")
counts = apps["Type"].value_counts()
# Pie chart
fig, ax = plt.subplots(figsize=(6,6))
wedges, texts, autotexts = ax.pie(
    counts.values,
    autopct='%1.1f%%',
    startangle=90,
    colors=["#66b3ff", "#99ff99", "#ffcc99"],
    pctdistance=0.7,
    textprops={'fontsize': 10}
)

labels = [f"{name} ({count})" for name, count in zip(counts.index, counts.values)]
ax.legend(wedges, labels, title="App Type", loc="center left", bbox_to_anchor=(1, 0, 0.5, 1))

plt.title("Free vs Paid Apps (Pie Chart)", fontsize=14)
plt.axis("equal")
plt.show()
```



Insight 3 — Ratings Distribution

This histogram illustrates how app ratings are distributed.
It helps identify whether ratings are generally high, low, or clustered around certain values, indicating overall user satisfaction.

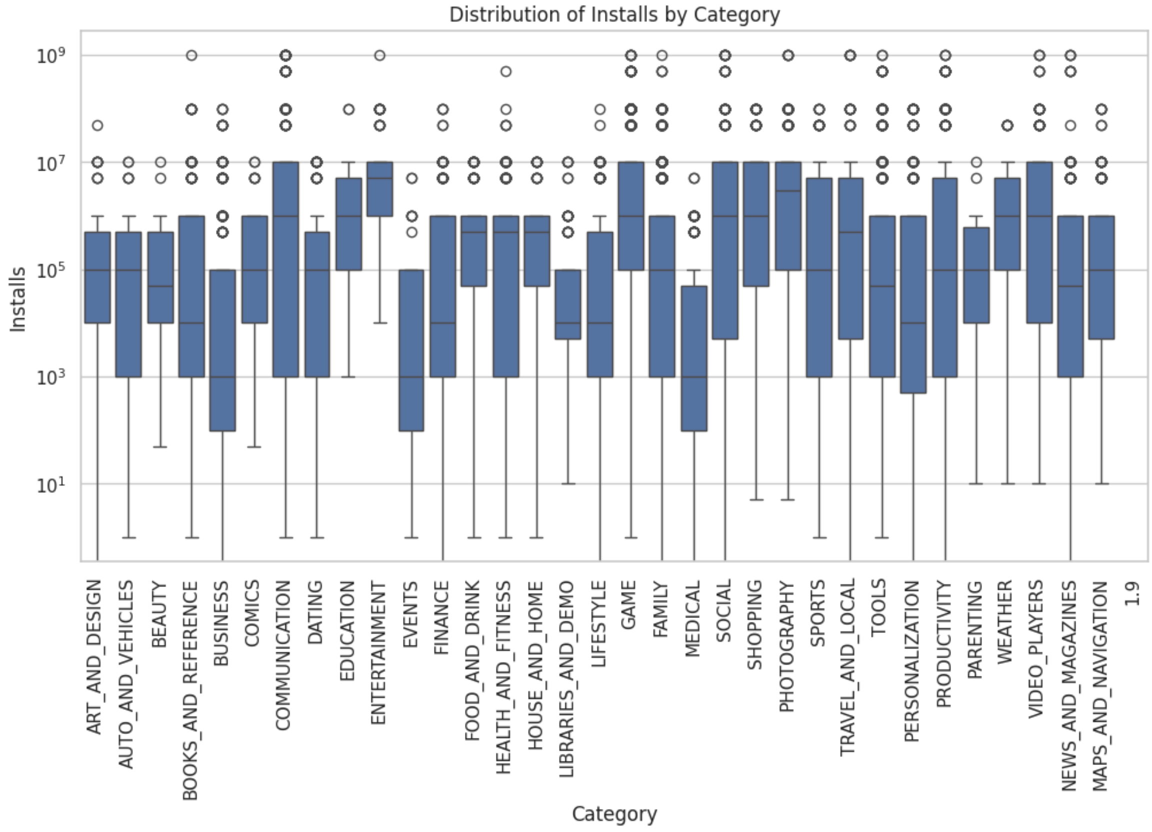
```
plt.figure(figsize=(8,5))
sns.histplot(apps["Rating"].dropna(), bins=20, kde=True, color="purple")
plt.title("Distribution of App Ratings")
plt.xlabel("Rating")
plt.show()
```



Insight 4 — Total Installs by Category

By summing installs per category, we can see which types of apps are most downloaded.
This indicates user interest and engagement across different app segments.

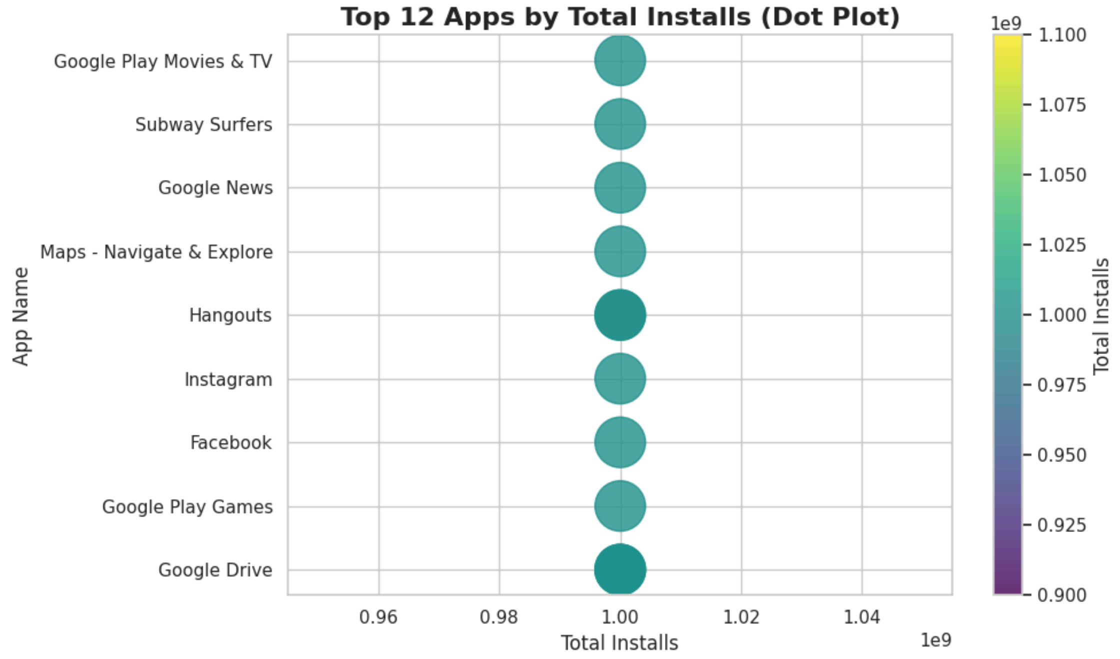
```
plt.figure(figsize=(12,6))
sns.boxplot(x="Category", y="Installs", data=apps)
plt.xticks(rotation=90)
plt.yscale("log")
plt.title("Distribution of Installs by Category")
plt.show()
```



Insight 5 — Most Popular Apps by Installs

This chart ranks individual apps by total installs.
It highlights leading apps in terms of user adoption and market penetration.

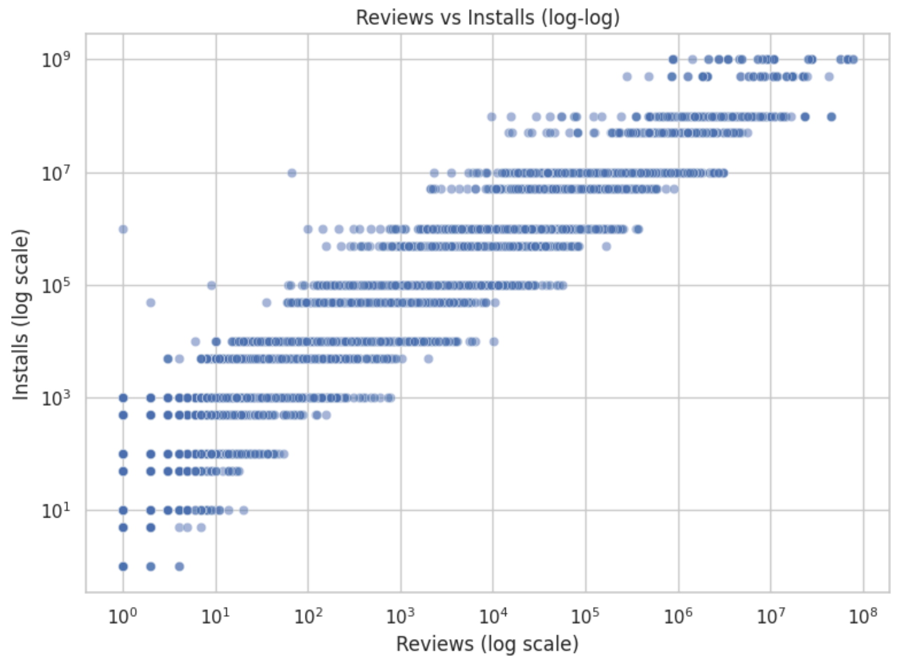
```
top_apps = apps.dropna(subset=["Installs"]).sort_values("Installs", ascending=False).head(12)

plt.figure(figsize=(10,6))
scatter = plt.scatter(
    x=top_apps["Installs"],
    y=top_apps["App"],
    s=top_apps["Installs"]/1e6,  # scale dot size
    c=top_apps["Installs"],
    cmap="viridis",
    alpha=0.8
)
plt.colorbar(scatter, label="Total Installs")
plt.xlabel("Total Installs", fontsize=12)
plt.ylabel("App Name", fontsize=12)
plt.title("Top 12 Apps by Total Installs (Dot Plot)", fontsize=16, weight='bold')
plt.tight_layout()
plt.show()
```



Insight 6 — Reviews vs Installs

A scatter plot showing the relationship between the number of reviews and total installs for each app.
Using a log-log scale allows us to see trends even when there is a large variance in values.
This helps evaluate whether highly installed apps also receive more user feedback.

```
# Scatter: Reviews vs Installs (log scale)
tmp = apps.dropna(subset=["Reviews","Installs"])
plt.figure(figsize=(8,6))
sns.scatterplot(x="Reviews", y="Installs", data=tmp, alpha=0.5)
plt.xscale("log")
plt.yscale("log")
plt.title("Reviews vs Installs (log-log)")
plt.xlabel("Reviews (log scale)")
plt.ylabel("Installs (log scale)")
plt.tight_layout()
plt.show()
```



Sentiment Analysis of User Reviews

Next, we analyze user feedback using **VADER Sentiment Analysis**.

Steps:

1. Compute a compound sentiment score for each review.
2. Classify reviews into **positive**, **neutral**, or **negative** categories.
3. Use these insights to understand user perception across apps and categories.

```
# Install VADER and compute simple sentiment scores (Colab)
# (You may see a short install message — that's normal)
!pip install -q vaderSentiment wordcloud

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

# Ensure reviews exist
if reviews.shape[0] == 0:
    print("No reviews file loaded — sentiment steps will be skipped.")
else:
    reviews["vader_compound"] = reviews["clean_review"].apply(lambda t: analyzer.polarity_scores(str(t))["compound"])
    def vlabel(c):
        if c > 0.05: return "positive"
        if c < -0.05: return "negative"
        return "neutral"
    reviews["vader_sentiment"] = reviews["vader_compound"].apply(vlabel)
    print("Computed VADER sentiment for reviews. Example:")
    display(reviews[["clean_review","vader_compound","vader_sentiment"]].head())
```

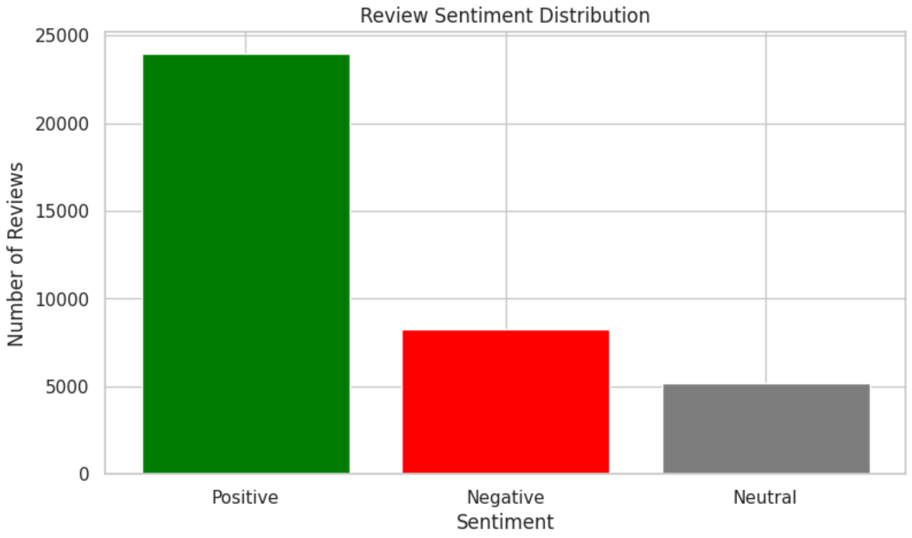
```
Computed VADER sentiment for reviews. Example:
```

	clean_review	vader_compound	vader_sentiment
0	I like eat delicious food. That's I'm cooking ...	0.9531	positive
1	This help eating healthy exercise regular basis	0.6597	positive
2	nan	0.0000	neutral
3	Works great especially going grocery store	0.6249	positive
4	Best idea us	0.6369	positive

Insight 7 — Distribution of Review Sentiment

This chart summarizes the proportion of positive, neutral, and negative reviews.
It provides a high-level view of user satisfaction across all apps.

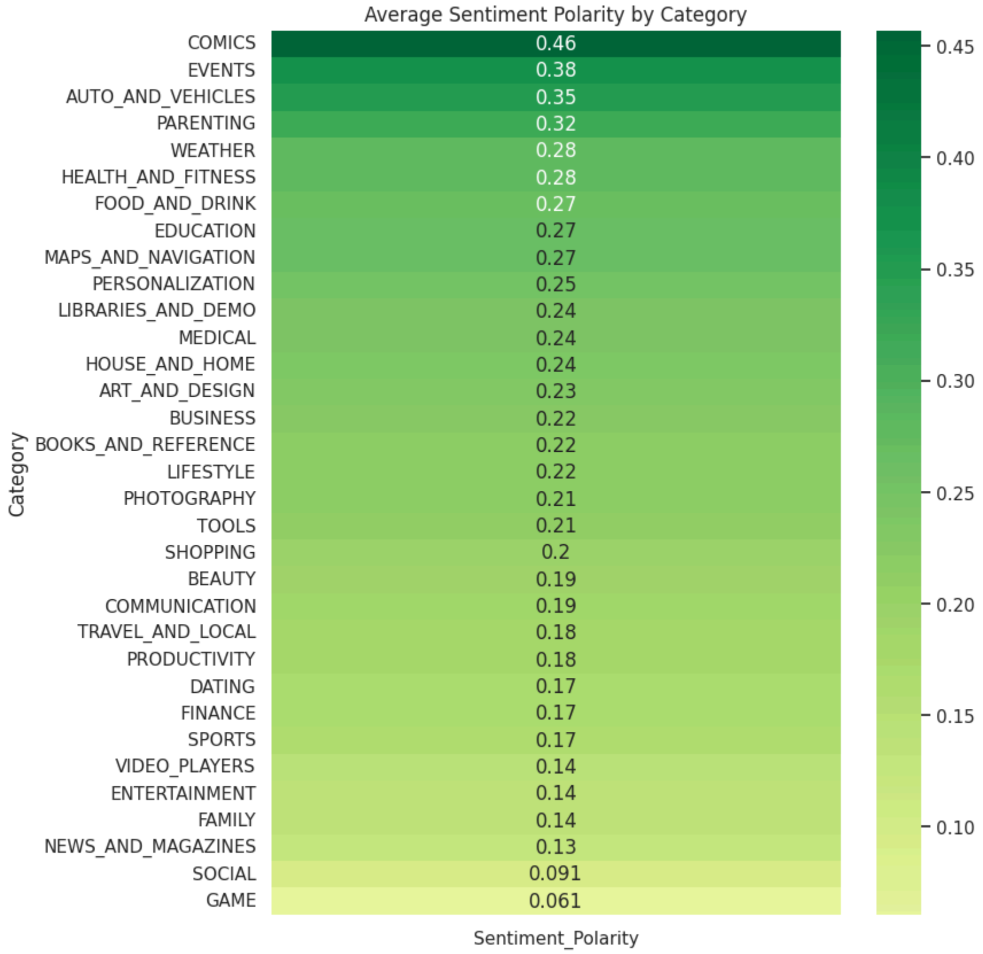
```
sent_counts = reviews["Sentiment"].value_counts()

colors = ["green","red","grey"]
plt.bar(sent_counts.index, sent_counts.values, color=colors)
plt.title("Review Sentiment Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Number of Reviews")
plt.show()
```



Insight 8 — Average Review Sentiment by Category

We calculate the average sentiment score for each category.
This identifies which categories tend to receive more positive or negative user feedback.

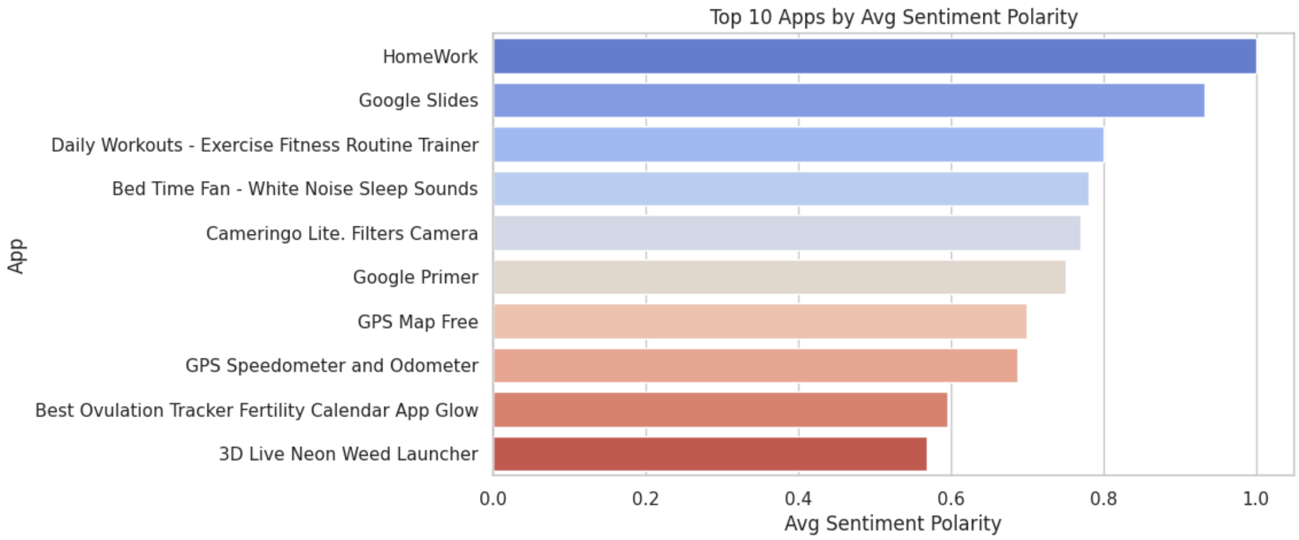
```
cat_sent = reviews.merge(apps[["App","Category"]], on="App", how="left")
cat_sent = cat_sent.groupby("Category")["Sentiment_Polarity"].mean().dropna().to_frame()

plt.figure(figsize=(8,10))
sns.heatmap(cat_sent.sort_values("Sentiment_Polarity", ascending=False), annot=True, cmap="RdYlGn", center=0)
plt.title("Average Sentiment Polarity by Category")
plt.show()
```



Insight 9 — Apps with Most Positive Reviews

This analysis highlights apps with the highest average sentiment, considering only apps with a sufficient number of reviews.
It helps identify apps that deliver the best user experience.

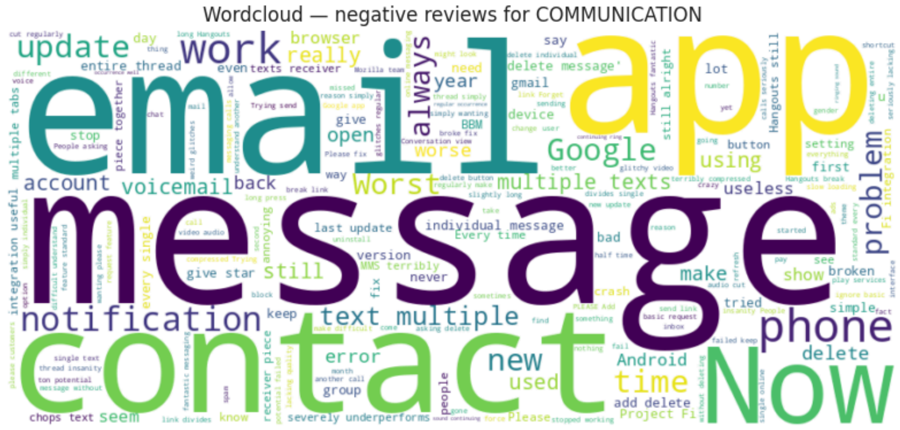
```
app_sent = reviews.groupby("App")["Sentiment_Polarity"].mean().dropna().sort_values(ascending=False).head(10)
sns.barplot(x=app_sent.values, y=app_sent.index, palette="coolwarm")
plt.title("Top 10 Apps by Avg Sentiment Polarity")
plt.xlabel("Avg Sentiment Polarity")
plt.show()
```



Insight 10 — Common Complaints in Negative Reviews

A wordcloud of negative reviews for a chosen category illustrates recurring user complaints.
This is useful for understanding areas of improvement for specific app types.

```
if reviews.shape[0] == 0:
    print("No reviews loaded. Skip wordcloud.")
else:
    from wordcloud import WordCloud, STOPWORDS
    chosen_cat = "COMMUNICATION"  # change this name if you want
    # join negative reviews for apps in that category
    neg = reviews[reviews["vader_sentiment"]=="negative"].merge(apps[["App","Category"]], on="App", how="left")
    texts = neg[neg["Category"]==chosen_cat]["clean_review"].dropna().astype(str)
    if len(texts) == 0:
        print(f"No negative reviews found for category '{chosen_cat}'. Try another category from apps['Category'].unique()")
    else:
        blob = " ".join(texts.values)[:200000]
        wc = WordCloud(width=900, height=400, background_color="white", stopwords=set(STOPWORDS)).generate(blob)
        plt.figure(figsize=(10,5))
        plt.imshow(wc, interpolation="bilinear")
        plt.axis("off")
        plt.title(f"Wordcloud — negative reviews for {chosen_cat}")
        plt.show()
```



Save Cleaned Data and Analysis Outputs

At the end of the analysis, save the following CSVs for future use or sharing:

1. `apps_cleaned.csv` — Cleaned app metadata
2. `reviews_with_vader.csv` — Reviews with computed VADER sentiment
3. `apps_with_sentiment.csv` — Apps merged with average sentiment

These files can be downloaded for reporting & presentation.

```
# Save outputs
try:
    apps.to_csv("apps_cleaned.csv", index=False)
    if reviews.shape[0]>0:
        reviews.to_csv("reviews_with_vader.csv", index=False)
    if 'apps_sent' in globals():
        apps_sent.to_csv("apps_with_sentiment.csv", index=False)
    print("Saved CSVs: apps_cleaned.csv", "reviews_with_vader.csv (if reviews exist)", "apps_with_sentiment.csv (if computed)")
except Exception as e:
    print("Error saving:", e)
```

```
Saved CSVs: apps_cleaned.csv reviews_with_vader.csv (if reviews exist) apps_with_sentiment.csv (if computed)
```

```
from google.colab import files

# 1️⃣ Save apps_cleaned.csv
try:
    apps.to_csv("apps_cleaned.csv", index=False)
    files.download("apps_cleaned.csv")
    print("Saved and downloaded: apps_cleaned.csv")
except Exception as e:
    print("Error saving apps_cleaned.csv:", e)

# 2️⃣ Save reviews_with_vader.csv
if 'reviews' in globals() and reviews.shape[0] > 0:
    try:
        reviews.to_csv("reviews_with_vader.csv", index=False)
        files.download("reviews_with_vader.csv")
        print("Saved and downloaded: reviews_with_vader.csv")
    except Exception as e:
        print("Error saving reviews_with_vader.csv:", e)
else:
    print("No reviews loaded. reviews_with_vader.csv not created.")

# 3️⃣ Save apps_with_sentiment.csv
if 'apps_sent' in globals():
    try:
        apps_sent.to_csv("apps_with_sentiment.csv", index=False)
        files.download("apps_with_sentiment.csv")
        print("Saved and downloaded: apps_with_sentiment.csv")
    except Exception as e:
        print("Error saving apps_with_sentiment.csv:", e)
else:
    print("App sentiment not computed. apps_with_sentiment.csv not created.")
```

```
Saved and downloaded: apps_cleaned.csv
Saved and downloaded: reviews_with_vader.csv
Saved and downloaded: apps_with_sentiment.csv
```